SPIDER | The Dutch Software Process Improvement Network

SPIder Koerier

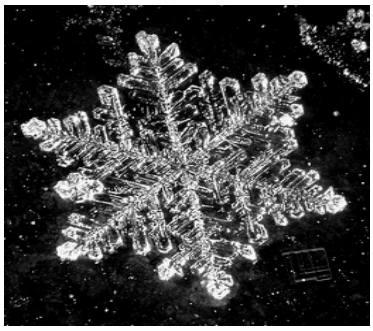
December 2009
Nummer 2009-4

Winter Editie 2009

www.st-SPIder.nl

Redactioneel

Buiten wordt het met de dag kouder: de winter is aangebroken in Nederland! Een mooi moment om de SPIder Koerier voor de open haard te gaan lezen. Dit is zeer zeker geen kwinkslag naar de cabaretier Herman Finkers. Want deze SPIder Koerier bevat een aantal interessante, wat langere artikelen over testen, integratie en configuratie management. Ik hoop dat deze Koerier wederom met veel plezier en aandacht wordt gelezen. Namens de redactie en het SPIder bestuur wens ik jullie prettige kerstdagen en een voorspoedig 2010 toe.



René Krikhaar

Inhoudsopgave

A methodology for testing (test) tools during LCM

Sander Koopman, ProRail

Abstract

This article presents a methodology for testing of tools during their Life Cycle Management (LCM).These tests mainly involve the testing of the implementation of the tool, but testing the tool itself cannot completely be ignored. Due to a number of factors the importance of a structured approach for testing of tools increases. As a first step, one should setup structured LCM for tools, with different types of events as a basis. For each event type, a standard approach should be described. As a second step, applicable quality attributes, test phases and test types have to be selected. Testing of tools requires some special test types and often an adapted test phase model. After that, one should define which test basis has to be used. Different from normal requirements based testing, software requirements and designs are not available when testing tools, so one needs to look abroad. Finally, one should select appropriate test techniques per test type.

Introduction

Tools are hot today. Tools for testing, test management, requirements management, change management and so on are increasingly used and tool vendors impress us with ever extending possibilities. However, making tools available and keeping them up and running isn't as easy as it looks. Tool architectures are quite complex and company networks often maintain a strict policy on system rights for tools. Knowledge about the tool is often limited or divided among different teams, which increases the risk on an erroneous installation. Such factors can cause a tool malfunctioning. And because a company's system environment often is very dynamic, it is not only necessary to test your tools initially during the tool selection and implementation, but also to define a test strategy for the entire lifecycle management (LCM) of the product. In this article I propose a test methodology for testing of tools, mainly derived from my experiences with tool LCM within a large financial company in the BeNeLux. Within this company, I was responsible for the LCM of test tools, especially of Quality Center, Quick Test Pro and Winrunner. On this tool set, we applied structured test management. Later on, we extended the test management approach also to tools of the Rational Suite (e.g. RequisitePro, ClearQuest). Until now, there isn't much existing literature on the testing of tools. Only Koomen et al. (2004) have written an approach to the testing of tools. Siteur (2005) has already pointed out that LCM of tools can require very intensive effort, however he hasn't worked out this into a methodology.

Differences between tools and tailor-built software

Before defining a suitable test strategy, it is important to realize that there are several differences between tools as a test object when compared with other software. A major difference is illustrated by **figure 1**. This figure sketches the position of different types of software solutions in a field of two dimensions: knowledge of software development process (X-axis) and knowledge of requirements (Y-axis). We can discriminate between the following situations:

a) Building software in-house
In this case, the complete software development process is executed in the company itself. Testers have insight in the contents of the requirements and in the development process. If the organization is mature in development and testing, testers could also have influence on the contents and management of the requirements, and on the development process. The requirements can be used as an input for the testing process.

b) Outsourcing of software development
In the case of outsourcing your software development, you don't have detailed knowledge of the development process of your sourcing partner. The development process of the sourcing partner is like a black box with requirements coming in, and software coming out. Requirements management is still in your company's hand and as a tester you can get detailed insight in requirements, and eventually influence their content or management. Testers will also be able to use the requirements as an input for the testing process.

c) Buying a tool
This is a very special case because the original requirements for the software are not known to you and you don't know anything about the development process of the tool. In most cases, a company will initially have a structured tool selection track in which a list of requirements is input for the track. These requirements also have to be used during a pilot or proof of concept with the tool. Obviously, the requirements for the selection track aren't equal to the requirements the tool vendor used for development of the tool. Further complicating factors for the test process are:

- As a tester you don't have any influence on the requirements and the functionality
- You don't know anything about development practices
- You don't know anything about the internal architecture and coding of the tool
- You aren't able to get defects repaired on a short term

The consequence of this is that classical requirements-based testing will be impossible for tools and that you have to define another strategy.

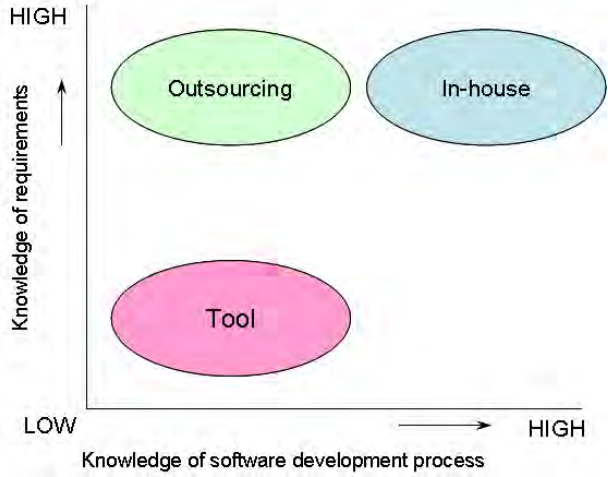


Figure 1: position of software solutions on the dimensions Knowledge of requirements and Knowledge of Software development process.

Why should you test tools?

More precisely, we should mainly speak about the testing of tool implementations, because we can assume that the tool itself should already be tested extensively by the vendor. Unfortunately we also know about situations in which functional regression occurred after having a patch installed, so you can't completely ignore testing the tool itself. But the motivation as described in this section is mainly applicable on the implementation of a tool, which delivers most problems in working with tooling.

There are several arguments which make it necessary to test a tool implementation. A first important argument is that many tools nowadays have a complex distributed technical architecture in which components are deployed on different infrastructure components. A typical simplified architecture is depicted in **figure 2**. This figure shows a tool with a major installation on a cluster of two web servers. Two types of client workstations connect to these web servers, eventually with additional software on the client. The web servers connect to a database server for storing the data. In reality the infrastructure will be much more complicated because components as authorization layers, different virtual servers on the web server and additional storage systems are not depicted in the figure. It would be very naive to suppose that such architecture should always work without any problems.

A second argument is the possibility offered by many tools to integrate with other tools. During this integration, often additional add-ins are loaded and tools are connected to each other which originally weren't designed to do so. This is especially the case if the tools are made by different vendors. This mechanism enlarges the chance of failure.

A third argument is the fact that many tools require extended rights on a workstation or database to function properly. For a database for example, a tool could require a special user for creating new tables or deleting tables. On workstations, tools often require modification of register settings or they have to write files to standard directories which are defined as read-only because of security policies. It is also possible that Internet settings need to be adapted which could also be in conflict with security. The modifications on client workstations are often not allowed to be done by end users, but are packaged in a client script. This client script manages the complete installation by copying the software and temporarily changing the system rights to enable register changes and file writing. Preparing such a client script is often a tricky task and errors after having installed a tool are not uncommon.

A fourth argument is that many companies nowadays are in a continuous stream of ICT fusions, integrations, in sourcing, outsourcing and so on, which often also leads to multiple tools or multiple versions of the same tool within one company. Often company management asks for consolidation of tooling which results in complex projects. Also rollout of tools in newly acquired parts of the organization can be a difficult job.

A fifth argument is the possibility for user customization provided by many tools. Often it is possible to write additional scripts for modifying screens or for changing

the workflow. Another customization issue could be the setting of specific parameters for example.
The five factors as mentioned here make it of utmost importance to define a thorough test strategy for your tooling to prevent that end users are confronted with blocking errors or gruesome messages.

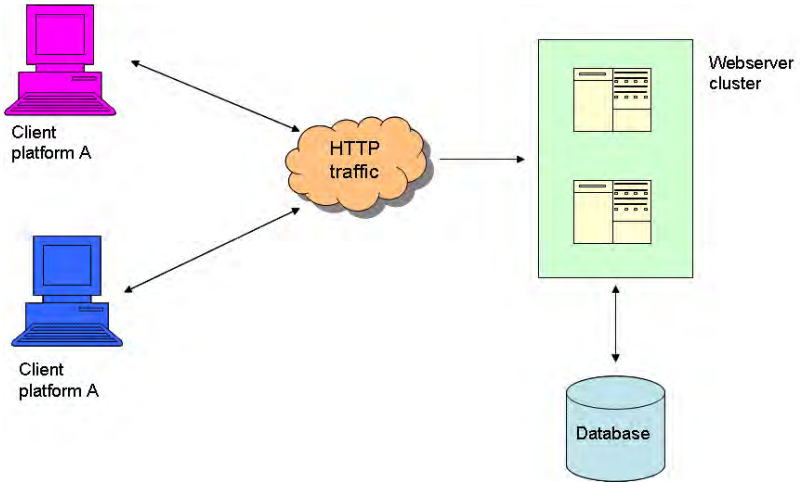


Figure 2: example sketch of the implementation of a tool with a distributed architecture. The environment consists of a cluster of two web servers, a coupled database and two types of client platforms.

Testing as a part of the tool LCM process

Testing of tools is part of the life cycle management (LCM) of the tooling. LCM involves all activities which keep the tooling and the infrastructure up to date. The relation between these concepts is depicted in **figure 3**. The tool is deployed on one or more infrastructure components. The tool as well as the infrastructure components are influenced by events. An event is a coherent set of activities to realize a goal of LCM.

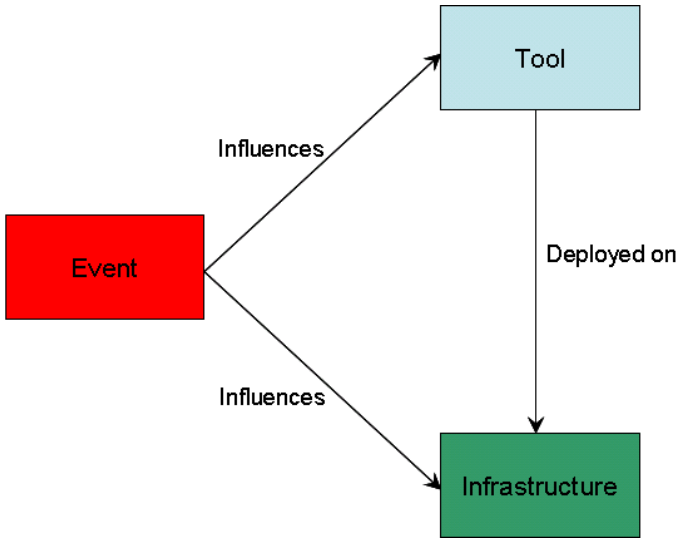


Figure 3: LCM concept model with tool, infrastructure and events.

There can be different types of events, varying from an update of the application to hardware replacement. A generic typology of LCM events is described in table 1. Every event should be detailed in a process flow and a description of activities.

LCM event	Description
Version upgrade	Implementation of a new version of a tool already in use
Patch upgrade	Implementation of a patch or update of a tool already in use
New package	Creation of a server or client package for a new tool
Package repair	Reparation of an existing server or client package because of malfunctioning
OS/DBMS upgrade	Upgrade of the operating system or database management system of one or more of the infrastructure components the tool is deployed on
Hardware migration	Replacement of infrastructure components
Tool integration	Setup of new integrations between tools or change of existing integrations

Table 1: generic LCM event types

Why is this important for testing? Each of these event types influences other parts of the tool and its infrastructure, and each event has it's own scale of impact on the tool environment. This means differences between the risks of event types, which also leads to different test approaches. Also for tool testing, the test process should not be limited to the final phases of events. The principle of the V-model for software development and testing can also be applied on LCM events. To do so, it is important to define a phased methodology for the LCM, in which test phases can be coupled to LCM event phases. A phase model for LCM events with coupled test process activities is depicted in table 2.

LCM event phase		Test process activities during this phase
Communication	Definition	-
	Impact assessment	Risk analysis, definition of test strategy
	Planning	Prepare test plan, test scenario and test cases
	Execution non-production	Execute tests
	Execution production	Pilot

Table 2: LCM event model with coupled test process activities

During the Definition phase the event is defined, and some preliminary research is done on the way the activities of the event should be planned and executed. Different alternatives for the event can also be studied during this phase (example: we are going to upgrade a tool, but which version will we choose?). Output of the definition phase is a report in which the event, the motivation for the event and the approach are described in headlines. During the next phase, the impact assessment phase, a detailed study has to be done on the effects of the event on the tool and the infrastructure. On the testing track it is time now to perform a risk analysis for the event, and to define the test strategy. In the Planning phase, all the gathered knowledge of the event has to be written down in a project plan and detailed test plan. Also the test cases and test scenarios have to be prepared within this phase. After the planning phase, during the Execution phase the event is

executed in one or more non-production environments and finally in the production environment. During the whole event track, it is of vital importance to communicate sufficient with your users. Most important questions for your users are: when is my tool unavailable, for how long will it be unavailable, will my data remain accessible, will I get a new user interface?
It is important to realize that it will be necessary for your organization to define a generic approach per type of event. This is because the different types of events require totally different activities, infrastructures and organizational disciplines. This has also consequences for the test strategy.

Test methodology for tools

Quality attributes

An important part of the test strategy is the selection of relevant quality attributes. For testing tools, we propose the quality attributes (ISO-9126) in table 3 as most important during LCM events.

Quality attribute	Sub characteristic	Aspects to check on
Functionality	-	-The most important functionalities of the tool
	Interoperability	-Integrations with other tools -Integrations between different infrastructure components
Efficiency	-	-Performance of the tool
Reliability	Maturity	-Frequency and category of errors experienced during use
	Recoverability	-Time and effort required for reinstallation or data recovery
Portability	Install ability	-Proper installation of the tool on servers and workstations -Possibility to install on other platforms or under other OS
	Conformance	-Compatibility of the tool with OS, DBMS and other relevant items

Table 3: most important quality attributes for tool LCM events.

It is clear that the quality attributes to check on during LCM events are different from those checked on during the tool selection. More precisely: a number of attributes important during the selection of a tool looses importance during the following LCM of the tool. This mechanism could cause a lack of attendance during the selection project for the quality attributes which are resident in both the selection project as well as the LCM. For example, *usability* is an extremely important quality attribute during the initial tool selection project, but it looses its importance during LCM, because users and application managers are already familiar with the tool then. Mainly *interoperability, maturity* and *installability* are undervalued during the selection project, but can deliver severe problems during later LCM events.

Test phases

For testing of tools the test phase model to be applied will be mainly dependent on the architecture of the tool. Table 4 depicts the necessary test phases. Just as for "normal" software the test process can involve the test phases System Test (ST), User Acceptance Test (UAT) and Production Acceptance Test (PAT). The Unit Test lacks because it isn't possible to test the code units inside the tool.

In general, with a more complex architecture one should also incorporate more test phases because a complex architecture involves more risks and in that case it could be beneficial to detect errors earlier. For a standalone architecture or client-server architecture a UAT phase should be sufficient. When the architecture is distributed, so also more complex, we recommend to include a ST. The ST and UAT phases should ideally have separate environments. For tooling, the PAT doesn't require a separate environment; one can use the UAT environment for it. An addition to the standard process is the pilot. Because it is impossible within large company networks to realize a fully equivalent test environment, it is strongly recommended to test your tools with a limited amount of users in the production environment before you release it to all users.

Architecture type	Test phases			
	ST	UAT	PAT	Pilot
Standalone	?	X		X
Client-Server	?	X		X
Distributed*	X	X	X	X

Table 4: required test phases and environments depending on architecture type of the tool.

*Distributed architecture: tool components on multiple servers, or with coupled databases.

Unless the architecture principle is a very useful guideline for defining your test phase model, there are some exceptions which have to be mentioned; that's why the cells for ST and UAT contain a question mark for Standalone and Client-Server. A ST phase could also be useful for these architecture types if it concerns a tool with many register changes or file system changes, with a known history of difficult installations or with many integrations with other products.

Test types

To be able to check on all the applicable quality attributes, different test types have to be performed. The following test types are possibly applicable for testing tools. Which types are definitely applicable for a certain tool depends on the architecture of the tool and on the type of LCM event:

1) Installation test

During this test you check if the tool installs properly on your infrastructure. This is not as evident as it looks. Mainly client installations can be very difficult because of security settings and a huge amount of other tools installed, which sometimes delivers conflicts between tools. Another nasty phenomenon can be the very different installation history of different workstations, which can deliver errors only on specific workstations while others haven't any problems.

For the installation test, it is important that you identify all the relevant environment variables and the values these variables can have in your production environment. As usually a large number of variables is involved, it is a challenging task to assemble representative test cases. Most important of the installation test is that you run the same tool installation on a number of environment configurations. An example from our practice to illustrate this kind of problems:

- Rational Clear Quest was installed and tested in a UAT environment with good results. However, the very first installation on a production workstation failed completely.
- Rational Requisite Pro was installed, configured and tested in a UAT environment without any problems. After rollout on the production platform it appeared that some users had problems and others had not, and that the error messages the users experienced were different per user. This was caused by different complex installation histories of the users' workstations.

2) Functional test

During this test you test if the main functionality of your tool works properly. For this test it is important that you have a clear picture of what should be the main functionalities for you. Make a difference between customizing features of the tool and real user features. Especially those functionalities which interact with system files or underwater with other tools are risky. Again some examples:

- After rollout of QTP it appeared that an update of the shared repository wasn't possible. This was due to security settings. As this functionality was vital for professional use of the tool, it took some time before QTP could be used as intended.
- After rollout of QTP it appeared multiple times that automated scripts for Internet Explorer applications suddenly delivered error messages and stopped running. This was caused by a security setting of Internet Explorer, which was automatically reset several times a day.

Unfortunately it is also necessary to perform functional regression tests on your tools to prevent unwanted consequences of your LCM events. Again an example:

- After an installation of HP Quick Test Pro it appeared that some functionalities of Rational Clear Quest didn't work anymore, which can be characterized as an unexpected interaction between tools.

In general, it would be feasible to automate the functional regression tests, because they comprise mostly very repetitive work. But the internal architecture of many tools often is so diverse that automated testing can become a difficult job.

3) Integration test

During this test you verify if the tool integrates well with the other tools as specified. Vendors often offer a wide range of integration possibilities, so it is important you get a clear picture of which integrations are important for your company. Strictly, you could also see the integration test as a separate test phase, but our experience pointed out that it is more convenient to execute the integration test as a test type within a ST or UAT phase. When setting up integration tests, important aspects to take into account are for example the vendors your tools are from (integrations between tools of one vendor often deliver less errors than between tools of multiple vendors), the direction of integration (unidirectional or bidirectional), the format of the data which will be exchanged, and the performance of the integration. Some examples to illustrate:

- Integration between HP Quality Center and Rational Clear Quest often runs difficult because of differences in format of data.
- After having implemented a new patch, fields exported from Quality Center to MS Excel were suddenly truncated after 255 positions.

4) Migration test

A migration test only involves the migration of data to other hardware or another format, because a migration of your tool to other hardware is already checked by installation tests and functional tests. Aspects to check on are the duration of the migration, and the accessibility, integrity and completeness of your data. Example:

> -When doing an upgrade, the vendor suggested that the data migration possibly could require a huge amount of time. Luckily, after a representative data migration in the UAT environment, it appeared that lapse time of the migration was much less than expected. So we could speed up the production migration with less downtime than initially foreseen.

5) Performance test

During a performance test you simulate the use of the tool by many concurrent users. This type of tests should only be executed on tools with a distributed architecture, if one expects a very high usage and if the tool fulfills a vital role for the organization.

How to arrange your test basis?

As stated before, normal requirements based testing isn't possible for tooling because we only have our own tool selection requirements. So you have to find other sources for test cases. The most useful sources are the user manuals. These could be the standard manuals of the vendor as well as manuals of your own company. From the manuals, you have to distill the most important functionalities. The descriptions in the user guide can act as use case-like descriptions of test cases. In addition to the user manuals, it could be useful to ask some key users for their most important functionalities. When you ask users, please be aware that you cover the different types of users (for example: admin users, experienced users, new users).

A second source for test cases can be the release notes of upgrades and patches of the vendor. The changes mentioned in these documents can be used as test cases to check if promised improvements are really delivered and work in your environment.

A third source is the list of historical and actual incidents in your tool set. If an error occurred anywhere in the past, it is recommended to write a test case for this error and to keep this case in your regression test set. This because it is very common that regression occurs, as well at the side of the vendor or somewhere in the tool implementation in your company. Following Graham & McKay (2008), this kind of testing can be described as "defect-based testing".

Finally, also the event documentation can act as a source. For example, an architecture document and project plan for a database migration can act as test basis for defining migration test scenarios. For complex LCM events, it is strongly

recommended to check your test types, test cases and scenarios with the tool vendor.

Applicable test techniques

Not all standard test techniques are useful for testing tools. In this section, I mention some techniques which could be useful, illustrated with some examples (sources: Graham &McKay (2008); Koomen et al. (2006)). It is important that during the Impact Assessment phase of your event also research will be done to define if, and if yes which, test techniques can be used. Test techniques mostly can be applied only for certain test types. This relationship is depicted in table 5.

	Installation test	Functional test	Integration test	Migration test
State transition testing			X	
Use case testing		X	X	
Syntactic testing			X	
All pairs testing	X			
Classification tree	X			
Data cycle testing		X		X

Table 5: possible use of test techniques per test type. Performance test is left out because there are specialized methodologies for this kind of testing.

1) State transition testing
This technique can be useful when testing integrations between tools. With this technique, the different states of the tools during the integration process can be described, as well as the actions which trigger the transitions between these states. Example: a custom-made tool which interfaced with HP Quality Center for exchanging defects was tested using State transition diagrams.

2) Use case testing
As stated before, descriptions of actions in the tool's user manuals or descriptions based on user experience can serve as use cases. Example: the way a Requisite Pro admin user creates a new database is documented and serves as a use case.

3) Syntactic testing
This can be useful mainly for integration testing. With syntactic testing you can check if the tools you want to integrate are sensitive to differences in data formats. This can have consequences for functional management of the tools, because you need conventions on how to enter data. Example: during tests of the integration between Clear Quest and Quality Center it appeared that Quality Center was case-insensitive and Clear Quest was case sensitive. As a consequence, user ID's in Quality Center needed to be changed.

4) All pairs testing
All pairs testing is a technique to reduce the number of test cases in a situation with many variables. This technique can be useful for installation tests. For this type of tests, often large numbers of system variables are involved. Applying multiple condition leads to an exploding number of test cases. When we assume that the different variables are independent, all pairs testing can be used to reduce the number of test cases strongly. Example: for an installation test, the following

variables could be of interest: operating system version, client environment, current version of tool on client, presence/absence of conflicting tools, browser version, installation type…

5) Classification tree
This is also a technique which is useful for situations with multiple variables which each could have multiple values. An advantage of this technique to all pairs testing is the graphical notation which gives quick insight in the numbers of variables and cases.

6) Data cycle testing
Data cycle testing is very useful to check if the different types of data within your tool can be created, read, updated and deleted. This is a very good technique to ensure that access rights to databases and file systems are sufficient. It can also be used to check the proper functioning of authorization groups within your tool. Example: when an admin user created a new Clear Quest database during the UAT, the creation process failed. This was due to wrong settings of the Oracle user.

Some tips for execution and finishing

Tool tests should ideally be executed by tool experts with assistance of key users. During execution of the tests, it is very important that all involved technical disciplines are available, to speed up the process of defect solving and retesting. Just as for testing of normal software, all test cases and scenarios should be archived for use during future LCM events.
It will also be necessary to develop a customized defect taxonomy and defect data model, with incorporation of the specific test phases, test types, tools and infrastructure components. And of course, after every event you should execute a thorough test process evaluation with all participants.

Acknowledgements

I would like to thank my former colleague René Thys for providing useful comments on an earlier version of this article.

Literature

Graham, B., & J. McKay (2008); “The Software Test Engineer’s Handbook”, Rockynook Computing.
Koomen, T., T. Vellekoop & J. van Vroonhoven (2004); “Testen & Quality Assurance (QA) van pakketimplementaties”. From: Koomen, T., and R. Baarda (ed.); TMap Test Topics, Sogeti.
Koomen, T., L. van der Aalst, B. Broekman & M. Vroon (2006); “Tmap Next voor resultaatgericht testen”, Uitgeverij Tutein Nolthenius.
Siteur, M.M. (2005); “Automate your testing-Sleep while you are working”, Academic Services.

Sander Koopman (1974) has a MSc degree in Physical Geography and is an ISEB practitioner certified test expert. From January, 2005 to October, 2008 he worked as a Test tool consultant for Fortis NL. Since October, 2008 he works for ProRail (Dutch railway managent company) as a Test analyst. He is also a member of the EXIN Professionals Group on Software Testing.

Integratie op basis van kwaliteit met branching strategie

Ruud Cox, Improve Quality Services BV

Stelt u zich eens voor: tijdens het bouwen van een huis is de timmerman met hout een deurkozijn aan het timmeren. Hij heeft net de eerste deurpost geplaatst, maar hij weet niet zeker of de deurpost waterpas staat omdat hij een gedeelte van zijn gereedschap is vergeten. Dat meet hij later nog wel een keer na. Ondertussen is de metselaar begonnen met het op hoogte brengen van de eerste muur. Omdat één deurpost al staat, is hij aan die kant begonnen. Hij neemt bakstenen van de stapel en legt ze stuk voor stuk, laag voor laag op elkaar. Hij maakt daarbij geen onderscheid tussen mooie en lelijke stenen. Om de vaart er goed in te houden, is de schilder ook van start gegaan. Hij had nog een gaatje in zijn agenda en kon alvast beginnen met de eerste deurpost. Doordat de timmerman nog aan het zagen is, komt er echter zaagsel op het natte schilderwerk waardoor het er lelijk uit komt te zien.

Als buitenstaander zouden we dit een hilarische manier van werken vinden. Veel van het werk zal opnieuw gedaan of hersteld moeten worden om een acceptabel eindresultaat te bereiken. Uiteindelijk zal het huis te laat opgeleverd worden en er zullen dan nog veel gebreken inzitten. Ten slotte zullen de hoge herstelkosten ervoor zorgen dat het budget wordt overschreden. Een ding is zeker: de koper van het huis zal niet tevreden zijn.

Ondanks het feit dat de hiervoor beschreven manier van werken hilarisch is, wordt bij het ontwikkelen van software vaak niet anders gedaan. Een softwareproduct wordt niet altijd in een optimale volgorde opgebouwd en ontwikkelaars integreren hun software tegelijkertijd in het eindproduct terwijl het ontwikkelwerk nog niet helemaal klaar is. De software is dan nog niet compleet en niet voldoende getest. De gevolgen zijn dan vergelijkbaar met die van het hierboven beschreven bouwvoorbeeld: het product komt te laat, de ontwikkelkosten overschrijden het budget en de kwaliteit is niet goed genoeg. Het is dus belangrijk om goed over een optimale volgorde van integreren na te denken in de vorm van een integratieplan. Om te voorkomen dat ontwikkelaars elkaar in de weg zitten bij het ontwikkelen kan in een configuratie management systeem voor elke feature een aparte branch aangemaakt worden. In een branching strategie staat welke branches er precies allemaal moeten komen. Door een feature pas te integreren als de kwaliteit goed genoeg is wordt voorkomen dat fouten onnodig in het product terecht komen. Met deze manier van werken is het wel mogelijk om een kwalitatief goed product op tijd af te hebben zonder dat het budget wordt overschreden.

Integratieplan

Bij het maken van een projectplanning wordt voornamelijk uitgegaan van beschikbare ontwikkelaars en testers, effort en doorlooptijd. Hierdoor ontstaat een plan waarbij taken in een soms onlogische volgorde of op verkeerde momenten tegelijkertijd moeten worden uitgevoerd. Denk hierbij aan de schilder die begint met schilderen terwijl de timmerman en de metselaar nog volop aan het werk zijn. Door aan het begin van een project stappen te definiëren waarin het product moet worden opgebouwd, wordt de volgorde van de uit te voeren taken wel logisch. Het

doel is namelijk dat na elke stap het product werkt en dat de kwaliteit goed genoeg is. Zo zal de schilder in een goed integratieplan pas helemaal op het einde aan de beurt komen omdat dit het beste moment is om te gaan schilderen. Eerder heeft geen zin want dan is de kans op beschadigingen veel te groot.

De gedefinieerde stappen worden vastgelegd in een zogenaamd integratieplan. De stappen worden dan ook wel integratiestappen genoemd. Zodra het integratieplan klaar is, wordt een projectplanning gemaakt. Als uit de projectplanning volgt dat de doorlooptijd te lang is of dat op sommige momenten in het project niet alle ontwikkelaars en testers werk hebben dan kan het integratieplan worden herzien. Er kunnen enkele iteratieslagen nodig zijn om tot een optimaal integratieplan en een projectplan te komen.

Branching strategie

Meestal bouwt een ontwikkelaar de software voor een feature stap voor stap op. Een feature kan een nieuwe functie voor de eindgebruiker zijn, maar het kan ook een redesign van een deel van het product zijn of zelfs een oplossing voor een fout. Na elk stapje test de ontwikkelaar of de software doet wat hij moet doen. Als er fouten in de software zitten dan lost de ontwikkelaar ze op en gaat hij verder met het volgende stapje.

Tijdens de ontwikkeling is de kans groot dat onderdelen van het product niet meer werken. Dat is logisch want de ontwikkeling is in volle gang. Als meerdere ontwikkelaars tegelijkertijd aan verschillende features werken, dan wordt het voor een ontwikkelaar heel moeilijk om te bepalen of de fout in zijn eigen software zit of dat het in een ander deel van de software zit. Als de metselaar constateert dat zijn muur scheef is, ligt dat dan aan de deurpost die mogelijk niet waterpas is of heeft hijzelf een fout gemaakt? Door features in isolatie te ontwikkelen kan deze verwarring voorkomen worden.

Software zit over het algemeen opgeslagen in een configuratie management systeem waarbij het eindproduct in de mainline is opgeslagen. Voor iedere feature wordt een feature branch gemaakt. Aan iedere feature kan tegelijkertijd gewerkt worden. Op deze manier ontstaat een geïsoleerde werkomgeving voor een ontwikkelaar. Hij heeft dan geen last van andere ontwikkelaars en andere ontwikkelaars hebben geen last van hem. Ook blijft het product in de mainline tijdens de ontwikkeling van de feature gewoon werken waardoor het mogelijk wordt om snel een release te maken waarin slechts een enkele fout is opgelost. Als de feature klaar is, dan wordt deze geïntegreerd in het product door de wijzigingen in de feature branch te mergen met de mainline. Zo zou de timmerman het deurkozijn in zijn eigen werkplaats moeten voorbereiden. Dan heeft hij geen last van de andere bouwvakkers en de andere bouwvakkers hebben geen last van hem. Als de timmerman het deurkozijn klaar heeft, dan pas wordt het deurkozijn geïntegreerd in het huis. Welke feature branches er aangemaakt moeten worden wordt afgeleid van het integratieplan. Het resultaat is de branching strategie voor het project.

Maar een branching strategie heeft nog meer voordelen: door een feature op een aparte feature branch te ontwikkelen heeft een project altijd de keus om de feature wel/niet of eerder/later te integreren. Als een deel van de software opnieuw is

geïmplementeerd met als doel een verbetering van de opstarttijd maar de beoogde winst wordt niet gerealiseerd, dan kan er voor gekozen worden om de feature niet te integreren. Als een projectmijlpaal onder druk komt te staan omdat een feature niet op tijd af is, dan kan ervoor gekozen worden om de mijlpaal te laten passeren en de feature later te integreren. Met een branching strategie wordt voorkomen dat ontwikkelaars last van elkaar hebben tijdens het ontwikkelen, is er altijd een werkend eindproduct en zijn projecten ook nog eens beter te beheersen omdat het werken met feature branches flexibiliteit oplevert.

Integratie op basis van kwaliteit

Als een fout in de software opgelost moet worden, dan kan dat maar beter zo vroeg mogelijk in het project gebeuren. Het oplossen van een fout die gevonden is door een klant is vele malen duurder dan een fout die gevonden is door een ontwikkelaar. Voor veel organisaties geld dan ook dat ze fouten zo vroeg mogelijk in een project willen vinden. Dat heeft tot gevolg dat er in een vroeg stadium getest moet worden. Ook al worden de testen netjes volgens plan uitgevoerd, dan kan het nog zo zijn dat het beter is om een feature nog even niet te integreren. Het is namelijk niet alleen belangrijk dat de feature compleet is en getest, maar het is net zo belangrijk dat de grootste of belangrijkste fouten in de feature zijn opgelost. Minder belangrijke fouten kunnen later.

Na de integratie van een feature gaan de systeemtesters aan de slag. Het is belangrijk dat zij het grootste deel van hun testen kunnen uitvoeren zonder al te veel problemen. Als dat niet zo is of er moet nog veel software aangepast worden om de reeds bekende problemen op te lossen, dan is het beter om de feature nog niet te integreren. Als na integratie nog de nodige software op de schop moet om de bekende problemen op te lossen, dan moet een groot deel van de testen opnieuw worden uitgevoerd.

Of een feature wel of niet kan worden geïntegreerd wordt besproken in een overdrachtsmeeting. Hier zijn alle betrokken ontwikkelaars en systeemtesters aanwezig. De meeting wordt voorgezeten door de integratie- en testmanager. Als alle criteria zijn getoetst dan wordt de go/no-go beslissing door hem genomen. Door een aantal kwaliteitscriteria te toetsen voordat een feature mag worden geïntegreerd in het product, ontstaat een quality gateway waarmee in een vroeg stadium fouten kunnen worden geweerd uit het product.

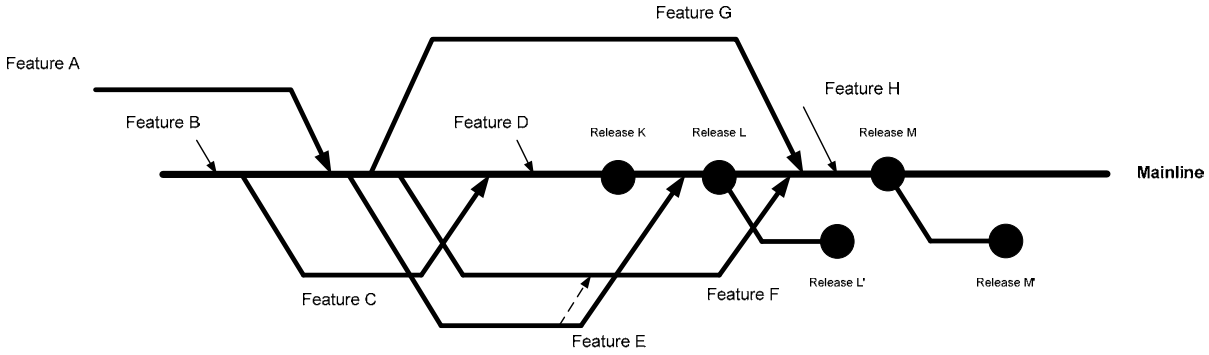
Een praktijkvoorbeeld

In een softwareproject bij de businessunit Cardio/Vascular van Philips Healthcare zijn de hiervoor beschreven verbeteringen stap voor stap ingevoerd. Aan het begin van het project werkten ongeveer tien ontwikkelaars tegelijkertijd aan hetzelfde product. Precies op de manier zoals beschreven in het bouwvoorbeeld met alle gevolgen van dien. Het kostte telkens veel tijd om het product stabiel te krijgen en ondanks het feit dat de ontwikkelaars veel tijd besteedden aan het testen, werden de meeste fouten toch gevonden door de systeemtesters.

Als eerste is het integratieplan onder de loep genomen. Hierbij zijn er meer en kleinere integratiestappen gedefinieerd en zijn alle taken in een optimale volgorde gezet. Dit ging niet in één keer goed want het was voor het project ook nog een leerproces. Al snel werd duidelijk dat bij het herzien van het integratieplan taken

werden geïdentificeerd die niet in de projectplanning stonden. De projectplanning was gemaakt op basis van features. Bij het identificeren van de taken om de feature te maken werden de taken die wel belangrijk zijn om het product te maken maar niet de feature, structureel vergeten. Zo moesten de testomgevingen met enige regelmaat van nieuwe software worden voorzien om de eigen software te kunnen testen. Een taak van vijf mandagen. Door zo'n taak te vergeten kan niet alleen de projectplanning behoorlijk overhoop gaan maar kan ook de voortgang in gevaar komen.

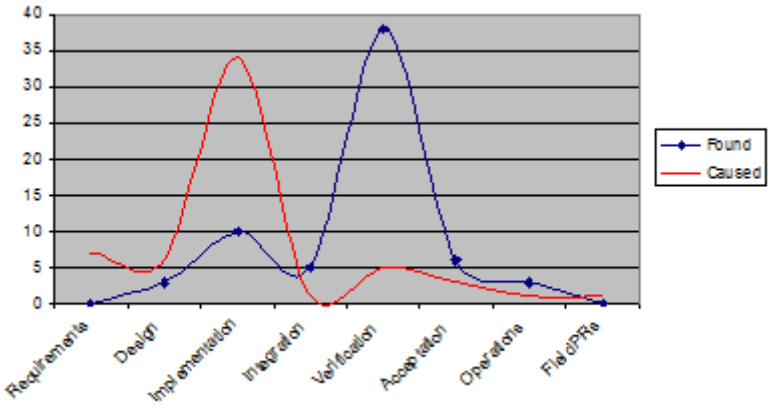
De tweede stap was het maken van een branching strategie. Hierdoor werd er voor gezorgd dat elke feature op een eigen feature branch kon worden ontwikkeld. Dit bracht rust in het project omdat ontwikkelaars elkaar niet telkens onnodig hoefde te helpen om te bepalen waarom een feature niet werkte.



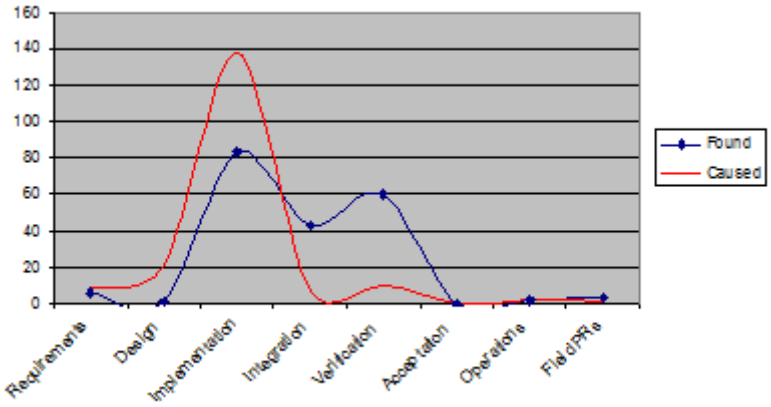
Figuur: Voorbeeld van een branching strategie

Ook zijn toen de eerste overdrachtsmeetings gehouden. De lijst van criteria werd in het begin nog niet erg strikt gehanteerd omdat dit weleens de nodige weerstand bij de ontwikkelaars zou kunnen oproepen. Van weerstand is echter geen sprake geweest. Ook zij zagen de voordelen van de gekozen manier van werken al snel in. Naarmate het project vorderde werd de lijst van criteria uitgebreid als dat nodig was en ook zijn de criteria strikter gehanteerd.

Verder is er volop gebruikgemaakt van de flexibiliteit van een branching strategie. Zo is het weleens voorgekomen dat een feature naar een volgende increment werd doorgeschoven. Het tegenovergestelde is ook gebeurd. Een feature die sneller af was dan gepland is wel eens eerder geleverd. Door het maken van een integratieplan en een branching strategie is het gelukt om het product op tijd af te hebben en binnen budget te blijven. Dat fouten eerder werden gevonden, wordt bevestigd door onderstaande grafieken. Hierin zien we de fase waarin fouten gevonden zijn en de fase waarin ze gemaakt zijn. De eerste grafiek geeft de stand na de eerste increment. De tweede grafiek geeft de stand na de vierde increment. Het is duidelijk te zien dat aan het einde van het project de fouten in een veel eerder stadium werden gevonden.



Figuur: Increment 1



Figuur: Increment 4

Conclusie

Functionaliteit, kwaliteit, kosten en tijd zijn de parameters waarop in een project gestuurd wordt. Met het maken van een integratieplan en een branching strategie, krijgt een project twee middelen in handen die helpen om het product op tijd af te hebben en de kosten binnen het budget te houden. Met het maken van een integratieplan wordt een product in de meest efficiënte volgorde opgebouwd. Met het maken van een branching strategie wordt het mogelijk om meevallers en tegenvallers in een project makkelijker op te vangen. Dit komt omdat het project flexibel om kan gaan met het moment waarop en feature geïntegreerd moet worden. Door te integreren op basis van kwaliteit worden hoge herstelkosten later in het project voorkomen. De kwaliteit van het eindproduct wordt met deze manier van werken niet echt beter. Om de kwaliteit van het eindproduct te verbeteren moet de teststrategie geanalyseerd en verbeterd worden.

De waarde van CM en gecertificeerde CM professionals

Inleiding

Dankzij de vele proces verbeteringen en nieuwe technologische ontwikkelingen in de IT, zijn we in staat grote en technisch complexe projecten uit te voeren. Met het stijgen van de project omvang, stijgt ook de noodzaak voor goede onderlinge

communicatie en beschikbare kennis en ervaring. Eenmaal opgeleverd, al dan niet binnen budget, tijd en met gewenste kwaliteit, is de beschikbaarheid van kennis en informatie over de producten en de samenhang er tussen, van eminent belang om verstoringen te voorkomen of snel te verhelpen. Hoewel het belang door management wordt onderkend, wordt het niet altijd in de praktijk toegepast.

Dit artikel beschrijft op basis van de praktijk een aantal aandachtsgebieden waarmee zowel projecten als lijnorganisaties kunnen profiteren van de waarde van Configuration Management (CM). Dit geldt voor elke methode van voortbrenging en onderhoud, maar met name voor trajecten waarbij gebruik gemaakt wordt van cyclische voortbrengingsmethoden, zoals RUP, agile development en extreme programming

Informatie ordening

Velen van u zullen de situatie herkennen waarin het succes mede afhankelijk is van de ‘goeroes’ die alles van uw project of systeem weten. Deze personen zijn zeer waardevol voor uw project, maar het heeft ook een keerzijde. Persoonlijke kennis is vergankelijk; met name als u gebruik maakt van externe resources. In deze moderne tijd blijven zelfs interne resources niet eeuwig bij uw project of organisatie.

Het op een juiste wijze borgen van afspraken, kennis en ervaring door middel van documentatie, leidt tot behoud van informatie en daarmee tot betere beslissingen. Het blijkt uit de praktijk dat de hoeveelheid vast te leggen informatie en het moment waarop dit wordt vastgelegd en ter beschikking gesteld wordt, kritische factoren in uw voortbrengingsproces zijn. De hoeveelheid vast te leggen informatie heeft een relatie naar uw risico’s. Alles vastleggen kan uiteindelijk leiden tot verlies van informatie; u ziet de bomen door het informatiebos niet meer. In beide extremen, alles of niets, heeft iedereen zijn eigen versie van de waarheid. Het moment waarop u informatie vastlegt en ter beschikking stelt, is met name bij een iteratieve aanpak van belang. Door belangrijke informatie bronnen, zoals requirements, ontwerpen en specificaties, te laat vast te leggen, loopt u risico’s op basis van verkeerde informatie te handelen. Informatie vastleggen dat nog veel aan verandering onderhevig is, kan extra werk tot gevolg leiden. Het niet vastleggen blijkt uit de praktijk geen optie te zijn.
Hoe compleet, geordend, en toegankelijk is bij u de informatie met betrekking tot uw ontwikkelde software systemen? En beschikt u over de juiste informatie op het moment dat u het nodig heeft?

De gevolgen van het ontbreken van informatie zijn niet altijd dramatisch als u uw ‘goeroes’ nog in de buurt heeft. Beslissingen op basis van incomplete analysegegevens, kunnen nog net op tijd bijgestuurd worden (de zgn ‘O ja’ momenten). Testers en gebruikers die gebruik maken van verouderde requirements, ontdekken dit nog net op tijd gedurende de diverse acceptatie testfasen. Onderhoudspersoneel dat ‘op goed geluk’ aanpassingen maakt, hebben soms dit geluk. Imagoschade is niet de enige drijfveer om uw informatie te managen; ook efficiënt omgaan met beschikbare resources is een drijfveer.

Project en Product informatie

De intensiteit van het voortbrenging en onderhoudstraject, wordt bepaald door beschikbare resources, budget en tijd. De uitkomst is een product met een bepaalde functionaliteit (scope) en kwaliteit. De projectmanagement vierkant wordt vaak gebruikt om de juiste balans tussen deze grootheden te vinden.
Tijd en geld worden uitgedrukt in datums en euro's, en zijn daardoor relatief eenvoudig te plannen en te monitoren. Met de grootheden scope en kwaliteit hebben we meer moeite. Bij het plannen en monitoren van deze laatste twee grootheden, zijn we sterk afhankelijk van de beschikbare informatie. Voorbeelden: functiepunten worden opgesteld op basis van verzamelde gegevens van afgeronde projecten, het testtraject wordt uitgevoerd op basis van requirements, specificaties en kwaliteitsnormen, het samenstellen en bijstellen van het product gebeurd op basis van architectuur, ontwerp en specificatie informatie.

We leven in een tijdperk van informatie en ons handelen is er afhankelijk van geworden. Des te vreemder is het dat we nog altijd moeten constateren dat de beschikbaarheid van correcte en complete informatie op het gewenste moment binnen de IT wereld te wensen overlaat. Zoals Johan Cruijff al memoreerde: "De aanval begint bij de doelman, en niet bij de middellijn of het doelgebied van de tegenstander". Ordening van informatie (Configuration Management) begint bij de start van uw project, en niet halverwege of bij overdracht aan beheer!

Configuratie Management

Veel disciplines resulteren in producten. Prince2 onderkent hierin sturende producten (bv programma en project informatie), en specifieke producten (bv systeeminformatie, software en testware). Het uiteindelijk te gebruiken product is een gevolg van eenduidig gedocumenteerde afspraken, variërend van contracten, vastgestelde eisen, ontwerpen en specificaties, te gebruiken omgevingen, tot informatie over de afzonderlijke delen van het eindproduct en de samenhang er tussen. Het identificeren, beheren en beheersen van deze documentatie en informatie is het werkterrein van de discipline Configuratie Management. In nagenoeg alle methoden heeft Configuratie Management om die reden een vaste plaats. In CMMI kan je niveau 2 niet bereiken als je dit niet op orde hebt. Zowel in Prince2 als in RUP wordt het als integraal onderdeel specifiek benoemd.
Configuratie Management is de discipline die vanaf het begin ordening in de te verwachte informatiechaos aanbrengt. Met deze informatieordening is functionaliteit en kwaliteit van het product beter voorspelbaar en resulteert in kwalitatief betere systemen in een kortere tijd. CM legt voor u het fundament om uw medewerkers optimaal hun eigen discipline optimaal en eenduidig uit te kunnen voeren. Dit geldt dus niet alleen voor software ontwikkelaars, maar ook voor ontwerpers, testers, business analisten, problemsolvers, teamleiders, projectmanagers en auditors.
In specifieke branches stelt onze maatschappij, via wetten en standaards, extra eisen aan traceerbaarheid van informatie. Voorbeelden zijn SOX voor financiële systemen, en de strenge regels gesteld door de FDA voor producten in de healthcare industrie. Ook hier biedt CM uitkomst; het biedt uw organisatie beschikbaarheid van de gewenste informatie.

Configuratie Management professionals

Iedere organisatie heeft Configuratie Management tot op een bepaald niveau geïmplementeerd. Het volwassenheidsniveau waarmee CM is geïmplementeerd, in combinatie met de personen die het uitvoeren, bepaalt het succes. Grote/middelgrote projecten uitvoeren zonder ervaren projectmanagers is geen sinecure; projecten zonder CM professionals evenmin. In de praktijk zien we dat bedrijven met specifieke CM professionals beter in staat zijn de CM discipline in te richten en uit te voeren. Bedrijven waar de CM activiteiten door rollen uit andere vakgebieden, zoals testers of programmeurs, uitgevoerd moeten worden, ontbreekt vaker de juiste informatie op het gewenste moment. Voor het goed uitvoeren van Configuration Management activiteiten blijkt de specifieke kennis en ervaring een voorwaarde te zijn.

Van veel van uw medewerkers verwacht u dat zij certificeringen hebben of behalen in hun vakgebied. Prince2 en IPMA worden veelgebruikt voor projectmanagers, T-map, ISEB en ISTQB voor uw testpersoneel, CMMI en ISO15504 voor proces assessoren, diverse certificeringen voor java ontwikkelaars of microsoft specialisten, en ITIL voor service managers. Welke certificeringen eist u van uw CM professionals? Hoe tonen zij hun professionaliteit aan? Op basis waarvan huurt u CM professionals in?

Configuration Management certificering

Met de erkenning van de waarde van Configuratie Management, groeit ook het aantal aanbieders op het gebied van CM certificering. Dit is een gezonde ontwikkeling, mits de certificeringen ook daadwerkelijk inhoud bieden. Het spreekt voor zich dat u bij uw zoektocht naar CM professionals, niet alleen kan laten leiden door zijn of haar behaalde certificeringen. Een CM gecertificeerd persoon heeft laten zien dat hij of zij het vakgebied begrijpt en geacht wordt de theorie naar de praktijk te vertalen. Het is de investering meer dan waard; de voorbeelden uit de praktijk bevestigen dit. Een professionele timmerman, zal ongetwijfeld kunnen stuken, maar u laat dit toch ook graag over aan een professionele stukadoor.
De belangrijkste spelers in relatie tot CM certificeringen, zijn:

- iNTCCM – public standard obv een Europees initiatief;
- CMII – voornamelijk gericht op hardware ontwikkeling;
- CMPIC – commerciële opleidingen, voornamelijk in Amerika;
- ITIL CM – voornamelijk gericht op service management.

iNTCCM

iNTCCM staat voor iNTernational Certification of Configuration Management professionals. De kracht en daarmee de voordelen van de iNTCCM certificering, is dat deze zich niet laten leiden door commercieel belang, maar het resultaat is van een samenwerkingsverband van zeer ervaren CM professionals onafhankelijk van bedrijven.
De iNTCCM Association heeft haar eerste CM kwaliteitsschema in 2007 gepubliceerd. De kenmerken hiervan zijn:

1. Het bevat best practices t.a.v. Configuration Management
2. Opgesteld door CM professionals uit de gehele wereld;
3. Biedt een standaard CM framework en terminologie;

4. Is public domain en is kosteloos beschikbaar via het internet (www.intccm.org).

Dit eerste schema bevat de basisprincipes van Configuratie Management, en wordt door commerciële organisaties gebruikt voor geaccrediteerde iNTCCM Foundation opleidingen. Een onafhankelijk examenbureau neemt de examens af; een opleiding vooraf is niet vereist, maar wordt sterk aangeraden.
De iNTCCM certificering wordt ondersteund door het Europese initiatief ECQA – the European Certification & Qualification Association.

René Schaap heeft meer dan 30 jaar IT ervaring, waarvan ruim de helft in het vakgebied Configuration Management. Hij is mede-initiatiefnemer en -oprichter van de iNTCCM Association en vervult momenteel de rol van president. Daarnaast is hij docent van de iNTCCM Foundation opleidingen en werkzaam als principal CM consultant bij Europese bedrijven.

Agile development and quality models: Do Agile on the one hand and ISO, SPICE and CMMI on the other hand match?

Met een volle zaal (67 aanmeldingen) zaten we op 30 november 2009 in de kantine van TomTom Automotive in Eindhoven te luisteren naar vier sprekers die ervaring hebben opgebouwd met Agile ontwikkelen. We vroegen ons af of het mogelijk is om Agile te combineren met de ‘oude’ kwaliteit modellen die tot stand zijn gekomen nog voordat Agile geboren was? Al heel snel was het antwoord duidelijk: JA, er is niets tegenstrijdig in de wereld van Agile ten opzichte van die van CMMI of ISO. Vier verschillende sprekers, vier verschillende ervaringen, vier verschillende lezingen; toch waren alle sprekers het met deze stelling eens.

Terugblik op de presentaties

Na de opening van de plenaire sessie door Jeroen Macke, voorzitter SPIder, kwam de eerste spreker - Lucien Jaegers van Philips Research MiPlaza – met een presentatie onder de provocerende titel ‘ISO 13485 en Agile,’ vrienden of vijanden’. Hij vertelde ons over de ervaringen van Philips Research met Agile Development in combinatie met de ISO13485 norm. Deze norm die een uitbreiding op ISO 9001 is, stelt eisen aan het kwaliteitssysteem dat gebruikt wordt voor de ontwikkeling en productie van medische apparaten. Vanwege de sterke focus op veiligheid en effectiviteit van medische apparatuur stelt deze norm strakke eisen aan de organisatie, zijn processen en aan de producten die gemaakt worden. Agile Software Development (ASD) staat daarentegen bekend om zijn vrijheid, en ook om zijn intrinsieke flexibiliteit. Toch heeft de praktijk bewezen dat de ASD kneedbaar genoeg is om zich ook aan ISO 13485 aan te passen en dat de ISO normen toch niet zo strak zijn als het in eerste instantie op lijkt.

De tweede spreker, André Heijstek van Improvementfocus, heeft ervoor gekozen om Agile naast CMMI te leggen. Onder het motto: ‘CMMI en Agile - synergie of dysergie?’ deed hij stap voor stap een vergelijking van de twee methoden. De aanname was dat de gebruikers van beide aanpakken zich niet bewust zijn van de

impliciete aannames in de methoden. André heeft deze impliciete aannames expliciet gemaakt en gaf suggesties voor een zinvolle synergie.

Na de pauze was het woord aan Martin Mermans van Philips Healthcare Informatics Infrastructure. In de industrie van de gezondheidszorg is het normaal om te moeten voldoen aan de Amerikaanse FDA norm (Food and Drug Administration) vertelde Martin. De FDA staat bekend om z'n strikte regels en om z'n krachtige middelen om deze norm af te dwingen. Philips Healthcare, zelf natuurlijk zeer kwaliteit bewust, had ook gekozen om intern andere standaarden zoals CMMI toe te passen. Toch is er bij Philips gekozen om Agile te ontwikkelen. Martin gaf een praktijkvoorbeeld dat heel duidelijk laat zien dat moderne ontwikkelmethoden heel goed samen kunnen gaan met allerlei verbetermodellen.

De laatste spreker, Rob Westgeest, sinds 2000 expert in eXtreme Programming, heeft deze problematiek van de andere kant benaderd. We gebruiken modellen of bepaalde technieken om onze software beter te maken. Rob heeft in zijn lezing teruggekeken naar de doelstellingen van software verbetering, naar wat we tot onze beschikking hebben om die doelstellingen te realiseren en hoe ver we kunnen komen. Agile lijkt duidelijk nog een lange toekomst te hebben en allerlei modeleen kunnen bruikbaar zijn zolang we softwareverbetering als ons doel houden.

Samenvattend

Zeer interessante lezingen, goede sfeer, leuk publiek, veel humor, heel veel voorbeelden uit eigen ervaringen van de sprekers en interessante vragen en opmerkingen na elke lezing. Zo kan je de laatste plenaire sessie van 2009 kernachtig typeren. Dankzij de gastvrijheid van TomTom hebben de deelnemers kunnen genieten van een lekker kopje soep en broodjes. Daarvoor wil ik namens SPIder TomTom heel erg bedanken.

Na de sessie hebben er nog aantal mensen de gelegenheid gebruikt om met elkaar na te praten, extra vragen aan de sprekers te stellen, eigen ervaringen met elkaar uit te wisselen. Om 21.00 uur was iedereen vertrokken.

Plenaire sessies 2010

We hopen volgend jaar opnieuw met een gevarieerd aanbod van inspirerende plenaire sessies te komen! Alle ideeën m.b.t. onderwerpen, sprekers en locaties zijn welkom en kunnen gemaild worden naar info@st-spider.nl .

Kasia Wiacek
SPIder bestuurslid

Q Society najaarsconferentie, 1 december 2009

Sleutels tot Succes

Soesterberg, 1 december 2009. De Society for Quality Professionals in ICT (Q Society) heeft op 1 december haar 8e conferentie gehouden over kwaliteit in IT. Het thema van deze najaarsconferentie was "Sleutels tot Succes". De Q Society is een initiatief van Mprise, AXYZ, KZA, LaQuSo, ps_testware, Qualityhouse, Sogeti, Stichting SPIder, Themis Consultancy en Valori.

Door Mario van Os en Marcel Brouwers

Hoe komt het dat het ene project een succes wordt, terwijl het andere volkomen faalt? Zijn er factoren te onderscheiden die de mate van succes van een project bepalen? En is het mogelijk om deze succesfactoren zelf toe te passen om je project tot een succes te maken?
Over deze vragen hebben circa 80 deelnemers van gedachte gewisseld aan de hand van de praktijkverhalen van een drietal sprekers die hun project, ondanks tegenslag, tot een succes wisten te maken. In de interactieve sessie die daarna volgde zijn we verder gaan ontdekken wat nu precies de sleutel tot succes was voor deze projecten en hoe je dit zelf zou kunnen toepassen.

Allereerst werden in een drietal presentaties voorbeelden gegeven van succesvolle uitvoering van projecten, waarbij door de sprekers werd ingegaan op de succesfactoren in hun omgeving.

Door Kees Stok en Martijn Kock werd een aantal sleutels tot succes bij KPN gepresenteerd. KPN heeft voor haar interne processen een complex IT landschap. Innovatie op dit landschap kent de klassieke uitdagingen: voorspelbare en bij voorkeur steeds kortere doorlooptijden, lagere kosten en hogere kwaliteit. Sinds begin 2009 is een performance verbeteringsinitiatief gestart waarbij de introductie van KPN Unified Process centraal staat. In deze voordracht is de balans opgemaakt: wat waren de sleutels tot de successen die al geboekt zijn? Welke uitdagingen zijn er nog?
In het bijzonder is aandacht besteed aan het onderwerp metrieken. Door Kees en Martijn werd de vraag gesteld "Is het vaststellen en meten van performance een sleutel tot succes?". Een werkgroep is hiermee aan de slag gegaan.

Vervolgens vertelde Carel Ilgen alles over het succes in een onmogelijk project. Een voor de meeste deelnemers herkenbaar project wordt geschetst: Een projectbudget en een doorlooptijd die niet realistisch kunnen zijn, een vage scope, 60 over elkaar struikelende leden van een projectteam, een samenwerkingsmodel dat niet functioneerde, wantrouwen tussen alle betrokken partijen, een lastig te bereiken hoofddoelstelling. Dit alles aangevuld met een aantal complexe

nevendoelstellingen, overtuiging bij de opdrachtgever dat de juiste weg bewandeld werd, stoppen of vertragen is geen optie. Verder waren er al zeven projectmanagers gesneuveld voordat Carel als achtste kwam. Nu, na een jaar, draait het project on scope, on schedule, on budget en is iedereen tevreden.
Wat was de sleutel tot succes?
Volgens Carel is het doorvoeren van een aantal basismaatregelen “Rust, Reinheid en Regelmaat” de kern om tot succes te komen.

Als derde en laatste presentator was Patrick Musters gevraagd om zijn ervaringen te vertellen over succesfactoren in een project vanuit een geheel andere optiek. Majoor Patrick Musters is F16 Test Piloot bij de Koninklijke Luchtmacht. De laatste jaren is Patrick tevens actief in het testen en evalueren van operationele systemen bij Defensie.
In zijn presentatie over de invoering van een systeem waarmee een poging gedaan zal worden om het gevaar van bermbommen te verminderen werd duidelijk dat kwaliteit een zeer belangrijke factor is om succes te bereiken. “Als kwaliteit een kwestie van leven of dood is!” krijgt een heel andere lading als het daadwerkelijk om mensenlevens gaat: “Failure is not an option!”
Volgens Patrick zijn de belangrijkste succesfactoren: Stay Focused, Teamwork, Accepteer 90% oplossing, Vertrouwen, Geduld en Money.

Na deze lezingen werden de 80 deelnemers in drie groepen verdeeld. Bij binnenkomst hadden ze daarvoor een kleur toegewezen gekregen die ze in de pauze konden omruilen met andere deelnemers. Het netwerk element, dat de Q Society zeer belangrijk vindt, werd daardoor gestimuleerd. Iedere groep ging met één van de sprekers, begeleid door een moderator vanuit de Q Society, naar een aparte zaal om het betreffende onderwerp verder te bespreken, uit te werken en de gestelde vragen te beantwoorden:

- Hoe komt het dat het ene project een succes wordt, terwijl het andere volkomen faalt?
- Zijn er factoren te onderscheiden die de mate van succes van een project bepalen?
- Is het mogelijk om deze succesfactoren zelf toe te passen om je eigen project tot een succes te maken?

De workshop sessie van KPN was een schoolvoorbeeld van maximaal resultaat in een minimale timebox. De door Martijn Kock en Kees Stok gepresenteerde set van metrics en het proces van implementatie werden van professionele kanttekeningen voorzien. Per groepje van drie is één advies uitgebracht aan Martijn Kock en Kees Stok. De adviezen van alle groepjes bij elkaar gaven een breed maar ook behoorlijk consistent inzicht in waar belangrijke “marge ter verbetering” en do’s en don’ts. De rode draad: het is vrij veel, kijk of je de hoeveelheid wat kunt terugbrengen, wellicht door wat metrics te combineren. En: pas op dat het geen al te IT-gedreven proces wordt. Zijn dit de metrics die de business wil zien?
Martijn Kock: “Ik ben blij met deze adviezen, het is duidelijk en we gaan hier zeker wat mee doen”. Kees Stok: “Bedankt voor de goede organisatie”.

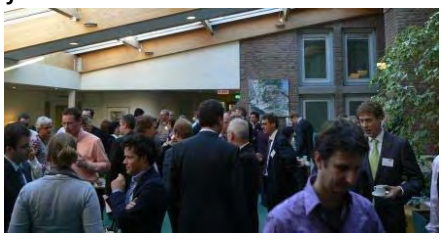
Tijdens de workshop sessie van Carel werd verder ingegaan op de succesfactoren binnen het geschetste project van Carel, maar ook praktijkvoorbeelden van de deelnemers. Een aantal aspecten die zijn besproken, zijn:

- Methoden & Technieken gebruiken is geen implementatie
- Minder mensen en minder teams geeft betere beheersbaarheid
- Communicatie & Samenwerking zijn de key succesfactoren
- Transparantie
- Juiste mensen op de juiste plek
- Multifunctioneel

In de workshop van Patrick Musters is geconcludeerd dat de door Patrick genoemde succesfactoren voornamelijk aan het eind van het project golden, tijdens de moeilijke fase doorontwikkelen of met slechte kwaliteit in productie. Belangrijke conclusie was dat succesfactoren afhankelijk zijn van de fase waarin een project zich bevindt, sommige factoren groeien tot succesfactoren en andere factoren gaan juist tegenwerken. Als voorbeeld kan genoemd worden de factor geld. Dit was aan het eind van het project een succesfactor, omdat het daardoor mogelijk was om twee weken extra door te werken en zelfs aan opschaling te doen. Geld daarentegen kan tijdens een project een risicofactor zijn in de zin van het creëren van steeds extra zaken omdat geld geen rol speelt.

De Q Society kijkt terug op een zeer succesvolle conferentie. Belangrijke succesfactoren voor een conferentie blijkt de samenhang van de verhalen (duidelijke rode draad), concrete opdrachten voor een workshop en entourage te zijn.

- Positief werd de rode draad gevonden; een quote uit de evaluatie: "*Goede sprekers/onderwerpen, zeker in relatie tot het thema Sleutels tot Succes*". In de voorbereiding op de conferentie is hier ook op gestuurd door de sprekers van te voren duidelijk te maken waar het over moest gaan, uitleggen wat je als Q Society verwacht te bereiken en de presentaties vooraf te reviewen.
- De sprekers gaven een opdracht aan of stelden een duidelijke vraag aan het publiek. Hierdoor kon in de workshop concreet gediscussieerd worden en konden de aanwezigen bewust voor een workshop kozen. "*Je leert het meest vanuit de praktijk, hoeft niet "eigen" praktijk te zijn*".
- De koffiecorner was erg gezellig en leende zich door zijn opstelling en indeling voor netwerken. De opstelling van een zaal en zijn omgeving is dus belangrijk voor de doelen. "*Goed om contact te leggen met de vakbroeders*".



Medio mei 2010 zal de volgende Q Society conferentie plaatsvinden. Wij hopen dan wederom op een goede opkomst en kwalitatieve discussies.

De Q Society is in 2005 opgericht onder auspiciën van Stichting SPIder. Q Society biedt een platform waarop functionarissen die betrokken zijn bij of verantwoordelijk voor kwaliteit in de ICT met elkaar van gedachten kunnen wisselen, kennis delen

en initiatieven ontplooien. Met als doel verhoging van de kwaliteit van informatiesystemen. De doelgroep waarop de Q Society zich richt zijn kwaliteitsprofessionals in de ICT-wereld.
De leden van de Q Society zijn gebruikers van ICT, aanbieders en afnemers van kwaliteitsdiensten en wetenschappers die willen bijdragen aan betere kwaliteit in de ICT.

Q Society ontplooit initiatieven om de community actief te voorzien van handvatten en kennis om de beoogde doelen te bereiken. Daartoe organiseert Q Society onder andere tweemaal per jaar een evenement voor kwaliteitsprofessionals. Het ene evenement richt zich vooral op informatieoverdracht, het andere heeft een workshop-karakter waarbij de deelnemers zelf aan de slag gaan

Meer informatie over de Q Society is te vinden op hun website www.qsociety.nl

De SPIder Organisatie

SPIder is de Nederlandse netwerkorganisatie voor SPI. SPIder organiseert jaarlijks een conferentie, minstens drie plenaire sessies met sprekers met variërende thema's. In aparte werkgroepen worden thema's bediscussieerd en verder uitgewerkt. De SPIder Koerier is het medium dat minstens viermaal per jaar verschijnt. Hierin kunnen lezers hun mening uiteenzetten en interessante ervaringen delen met SPIder leden. SPIder is een stichting, non-profit organisatie, welke wordt bestuurd door vrijwilligers.

Lidmaatschap van SPIder is gratis, en alle activiteiten met uitzondering van de SPI conferentie zijn gratis toegankelijk. Dit wordt mede mogelijk gemaakt door onze reguliere sponsors. Dus dank voor de bijdrage van Philips, Sogeti, KZA, Logica en DNV-Cibit!

Ook kent SPIder donateurs, zowel bedrijfsmatig als individueel. Donateurs hebben bij ons een streepje voor, en krijgen extra voordeel op de activiteiten van SPIder en van onze zusterverenigingen. Wil je SPIder ook steunen, meld je dan aan bij het SPIder secretariaat.

Het SPIder bestuur bestaat uit de volgende personen:

- Jeroen Macke, voorzitter
- Niek Pluijmert, penningmeester, Q Society
- Martin Muller, strategie, donateurs
- Kasia Wiacek, plenaire sessies
- Wil Leeuwis, Website, SPIder conferentie
- René Krikhaar, SPIder conferentie, SPIder Koerier

Informatie over SPIder is te vinden op de website: www.st-SPIder.nl

Voor reacties en bijdragen op de ***SPIder website*** kunt u zich richten tot:
Redactie SPIder web, Wil Leeuwis
E-mail: w.leeuwis@gmail.com

Colofon

De inhoud van de SPIder koerier wordt verzorgd door het SPIder netwerk. Dat betekent dat de redactie met plezier artikelen ontvangt voor publicatie. Goede ideeën om op andere wijze zinvolle inhoud te geven aan de Koerier zijn welkom: een speciale reeks van artikelen rondom een bepaald thema, een discussie via de Koerier in de vorm van meningen en opinies of een column. Uw ideeën zijn welkom, we kunnen ze bespreken tijdens de conferentie, per mail of telefoon.

De SPIder Koerier wordt gemaakt onder auspiciën van SPIder door René Krikhaar en Cantrijn Secretariaten. Voor artikelen, mededelingen, reacties en vragen m.b.t. de ***SPIder Koerier*** kunt u zich wenden tot de schrijvers of

SPIder Koerier
E-mail: koerier@st-SPIder.nl

Volgende deadline van de SPIder koerier is 15 maart 2010.